AMENDMENTS TO THE SPECIFICATION

Page 8, line 5:

FIGURE 1A is a block diagram illustrating an exemplary architecture configuration in which the claimed invention is operable.

Page 8, line 7:

FIGURE ~~1A~~1B is a flow diagram illustrating the steps undertaken by the architecture configuration of ~~FIGURE 1~~ FIGURE 1A to enable internodal messaging of unrestricted length.

Page 8, line 10:

~~FIGURE 2~~ FIGURE 2A illustrates the layout of the Input Command register in a preferred embodiment enabling the claimed invention.

Page 8, line 12:

FIGURE ~~2A~~ 2B illustrates the layout of the Operation Context register in a preferred embodiment enabling the claimed invention.

Page 10, line 24:

With reference to ~~FIGURE 1~~ FIGURE 1A, therefore, there is illustrated an exemplary architecture configuration in which the claimed invention is operable, in which EPAC (Excalibur Processor Agent Chip) 150 includes CPU Input Registers 151 and 152 (corresponding to CPU 0 and CPU 1 respectively) available to Message/Copy State Machine 153. As part of data movement operations, and particularly advantageous in the notification phase of data movement, Message/Copy State Machine 153 sends information to CPU Status Queues 154 and 155. CPU Status Queues 154 and 155 again correspond to CPU 0 and CPU 1 respectively, and are also resident on EPAC 150.

Page 11, line 9:

Under the claimed invention, data movement operations are enabled almost exclusively using hardware as illustrated on ~~FIGURE 1~~ FIGURE 1A. Traditionally, phases or aspects of data movement such as memory allocation and notification are enabled by software. As will be described below with respect, for example, to messaging, these phases or aspects are enabled by hardware under the invention.

Page 11, line 14:

First, however, it is useful to visualize the hardware components of ~~FIGURE 1~~ FIGURE 1A in a larger perspective reflecting a multi-processor, multi-node environment. With reference to FIGURE 14, therefore, there is illustrated an exemplary architecture configuration showing the interaction of EPAC 150 and EMAC 160 with other hardware in such an environment. EPACs 150 are attached to processors 1401 (advantageously 2 processors per EPAC). As illustrated on FIGURE 14, an I/O subsystem may also be attached to each EPAC using unidirectional paths. Additionally, EPACs 150 are attached to core logic bus 1402, through which EPACs 150 may interface other computer system components such as Electrically Erasable Program Read Only Memory ("EEPROM"), Static Random Access Memory ("SRAM"), real time clock, RS-232 and ethernet. Core logic bus 1402 also enables processors 1401 to write to Control and Status Registers ("CSRs") which are accessed using the bus to initialize and configure cross bar gate arrays ("ERACs") 1403.

Page 13, line 15:

Turning now to FIGURE ~~1A~~ 1B, messaging begins at block 101, in which the source processor formats the message to be sent. The message may be of any length. The actual formatting of the message is done by software and is performed using system memory.

Page 13, line 19:

In block 102, the source processor issues a command to commence the messaging operation. This command is issued to the hardware on the local node by writing to input command registers on the local processor agent (EPAC 150 on ~~FIGURES 1~~ FIGURES 1A and 14).

Page 13, line 23:

In block 103, a Message/Copy State Machine within the EPAC sends a transaction to the memory access controller (EMAC 160 on ~~FIGURES 1~~ FIGURES 1A and 14) on the receiving node. This transaction is directed to the message allocation state machine within the receiving EMAC (item 161 on ~~FIGURE 1~~ FIGURE 1A), and includes the source address of the message in memory and the size of the message.

Page 14, line 1:

In block 104, the EMAC message allocation state machine allocates memory on the receiving node from a pool of memory under its control. The amount of memory allocated corresponds to the size of the message that has to be stored therein. The EMAC then sends a transaction back to the message/copy state machine (item 153 on ~~FIGURE 1~~ FIGURE 1A) within the source EPAC, where that message includes the destination address that has been allocated by the receiving EMAC in receiving node memory (block 105).

Page 14, line 15:

Once the entire message has been successfully copied, the source node sends a completion status to the receiving node (block 107). The sending of the completion status is enabled by the Message/Copy State Machine on the source EPAC sending a completion status transaction to the EMAC on the receiving node. The EMAC then enqueues the completion status transaction in a queue for that node (item 162 on ~~FIGURE 1~~ FIGURE 1A). When an empty queue enqueues a completion status (i.e. the queue transitions from empty to non-empty), an interrupt is sent to a processor local to that node so that software can process the received message (block 107A). An interrupt is optionally also sent to the sending processor (block 107B). In this way, the sending processor will know that the messaging operation has completed and that it can proceed accordingly.

Page 15, line 5:

With reference again to ~~FIGURE 1~~ FIGURE 1A, therefore, messaging will now be described by reference to specific hardware components. EPAC 150 is resident at the source or sending node from which a message is desired to be sent. EMAC 160 is resident on the receiving or destination node for the message. As noted above, EPAC 150 comprises CPU Input Registers 151 and 152 available to be programmed by two separate processors CPU 0 and CPU 1 respectively. EPAC 150 further comprises Message/Copy State Machine 153, and CPU status queues 154 and 155 available to condition processors CPU 0 and CPU 1 respectively. EMAC 160 comprises message allocation state machine 161 and message completion status queue 162.

Page 15, line 28:

The execution of a messaging operation is divided into three phases. The first phase is determining the destination address for the message. If the current operation is a copy operation then this phase of execution is skipped. As shown in ~~FIGURE 1~~ FIGURE 1A, the destination address is determined by sending transaction 170 to EMAC 160 on the destination node. EMAC 160 performs a memory allocation operation and responds with a destination node memory address (transaction 175).

Page 16, line 16:

An operation status queue 154 and 155 is associated with each processor physically connected to EPAC 150. On ~~FIGURE 1~~ FIGURE 1A, these processors are CPU 0 and CPU 1. Status resulting from the completion of a message or copy operation is placed in one of the two status queues. An operation started by CPU 0 when complete will write status to its assigned status queue 154. Similarly, CPU 1's status will be written to status queue 155. Each status queue is three entries deep to provide status space for the Input Register and Message/Copy State Machine stages.

Page 17, line 1:

~~FIGURES 2~~ FIGURES 2A through 11 describe the CSRs (Control and Status Registers) which are required to control the messaging and data copy hardware. All CSRs reside either in EPAC 150 or EMAC 160 as illustrated on ~~FIGURE 1~~ FIGURE 1A. CSRs include:

- Input Command register
- Operation Context register
- Source and Destination Physical Page Frame registers
- Source and Destination Offset registers
- Operation Status register
- Message Reception Area Configuration registers

- Message Reception Area Offset registers
- Message Completion Queue Configuration registers
- Message Completion Queue Offset registers
- Memory Allocation address
- Message Completion Enqueue address
- Message Completion Dequeue address

Page 17, line 23:

The format of the Input Command register is shown in ~~FIGURE 2~~ FIGURE 2A. The fields of the Input Command register are defined as follows:

Page 21, line 8:

The format of the Operation Context register is shown in ~~FIGURE 2A~~ FIGURE 2B. The fields of the CSR Operation Context register are defined as follows:

Page 24, line 4:

Multiple Operation Status Queue registers are provided on each EPAC 150, one for each processor attached thereto. In the embodiment illustrated on ~~FIGURE 1~~ FIGURE 1A, there are two processors CPU 0 and CPU 1, and so two Operation Status Queue registers are provided. Status is inserted in the status queue in the order that the operations complete. Note that this order may not be the order that the operations were issued if errors have occurred.

Page 45, line 7:

As noted above, a TLB purge is a purge of the Translation Lookaside Buffer, indicating to a processor that there has been a change in virtual page to physical page translation. Referring to FIGURE 15, the TLB purge alarm mechanism is armed in block 1501 prior to starting the data copy operation. In the exemplary architecture configuration described above with reference to ~~FIGURES 1~~ FIGURES 1A and 14, this is done by writing to the Operation Context CSR, described above in association with ~~FIGURE 2A~~ FIGURE 2B. It will be noted in reference to ~~FIGURE 2A~~ FIGURE 2B that the Operation Context CSR is illustrated having an armed bit and a triggered bit. Accordingly, setting the armed bit in that Operation Context CSR instructs the hardware that an operation controlled by the TLB purge alarm is being set up.

Page 45, line 19:

The TLB purge alarm mechanism then starts monitoring for TLB purges in block 1502. While this monitoring continues, virtual to physical translation commences in block 1503 according to the currently-prescribed mapping. This translation is done for both the source and the destination locations in the data copy operation specified. Once this translation is complete, the data mover registers are set up in preparation for the copy operation (block 1504), and the "ready" bit in the Input Command CSR (refer back to ~~FIGURE 2~~ FIGURE 2A) is set (block 1505). The setting of the Input Command CSR "ready" bit in block 1505 informs the hardware that all preparations for actual data movement have now been completed.